# اصول پردازش تصویر

نیم‌سال اول ۱۳۹۹-۱۴۰۰

مدرس: دکتر مصطفی کمالی تبریزی

دانشکده علوم ریاضی

**تمرین سری سوم ــ سوال سوم**

نام و نام‌خانوادگی: سیدعلیرضا خادم

شماره دانشجویی: ۹۷۱۰۰۳۹۸

زمان حدودی اجرا برای ۲۰۴۸ سگمنت و تصویر با سایز اصلی : ۱۵ ثانیه

## موارد لازم.

برای اجرا لازم است تا تصویر slic.jpg در مسیر /EX3_Q2/images قرار داشته باشد. همچنین در پیاده‌سازی این سوال از کتابخانه‌های numpy ، skimage و cv2 استفاده شده است که قبل از اجرا بایستی این کتابخانه‌ها روی سیستم شما نصب باشد.

## روند کلی حل.

ایده کلی الگوریتم slic بسیار مشابه الگوریتم Turbo pixel است و به این صورت است که در ابتدا تعداد کلاسترهایی که میخواهیم در نهایت داشته باشیم را در نظر می‌گیریم و مرکز این کلاسترها را به صورت یونیفرم در تصویر پخش می‌کنیم. قبل از هر کاری در ابتدا هر یک از این cluster center ها در یک پنجره‌ای به ابعاد ۱۱ در ۱۱ به مرکزیت خودشان جابه‌جا می‌شوند تا در مکانی از این پنجره که کمترین گرادیان را دارد قرار بگیرند. بعد پیکسل‌های اطراف هر کدام از این کلاستر سنترها بر اساس معیارهایی که در روابط زیر مشاهده می‌کنید به کلاستر سنترها اختصاص می‌دهیم.

$$
\begin{aligned}
d_{lab} &= \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \\
d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\
D_s &= d_{lab} + \frac{m}{S} d_{xy} \ ,
\end{aligned}
$$

در رابطه بالا S فاصله دو کلاستر سنتر بعد از توزیع یکنواخت آنها روی تصویر است و ما در این سوال m را مساوی ۲۲ در نظر گرفته‌ایم.

پیاده‌سازی‌ای که در این کد صورت گرفته است تا حدی زیادی مشابه توضیحاتی است که در صورت سوال آمده است اما تفاوت‌هایی هم با آن دارد که این تفاوت‌ها عبارت اند از : در روشی که در صورت سوال آمده است یک آستانه برای

اختصاص دادنِ یک پیکسل به یک کلاستر در نظر گرفته شده است به طوری که اگر $D_s = d_{lab} + \frac{22}{S} d_{xy}$ از آن آستانه کمتر باشد به آن کلاستر سنتر اختصاص داده می‌شود. در این روش امکان این که تعدادی پیکسل در نتیجه نهایی به هیچ کلاستری اختصاص داده نشوند یا به چند کلاستر اختصاص داده شوند وجود دارد، اما پیاده سازی‌ای که در این کد انجام شده است به این صورت است که کلاستر سنترها را از گوشه چپ بالا پیمایش میکنم و در هر گام پیمایش پنجره‌ای به ابعاد $2S * 2S$ پیرامون کلاستر سنتر در نظر می‌گیریم و مقدار $D_s$ را برای هر پیکسل در این پنجره و کلاستر سنتری که در حال حاضر روی آن قرار داریم محاسبه میکنیم. آرایه data را به ابعاد $image\_height * image\_width * 2$ در نظر می‌گیریم به گونه‌ای که $data[x, y, 0]$ نشان میدهد که پیکسل $x, y$ آخرین بار به کدام کلاستر اختصاص داده شده است و $data[x, y, 1]$ نشان می‌دهد $D_s$ دراختصاص این پیکسل به $data[x, y, 0]$ چقدر بوده است. در ابتدا همه داریه‌های $data$ را با مقدار $np.inf$ مقداردهی می‌کنیم. در هر گام از پیمایش کلاسترها اگر پیکسل $D_s$ ای که دارد از $data[x, y, 1]$ کمتر باشد به این معنی است که این پیکسل بیشتر به این کلاستر نزدیک است تا $data[x, y, 0]$ ؛ در نتیجه مقادیر $data[x, y, 0]$ و $data[x, y, 1]$ را آپدیت می‌کنیم. هر بار که یک پیکسل به یک کلاستر اختصاص داده می‌شود، مقدار کلاستر سنتر متناظر را آپدیت می‌کنیم.

# توضیح کد.

برنامه در مجموع حاوی ۲ فایل با فرمت py. می‌باشد که توضیحات هر فایل در پایین آمده است.

### ○ utilities.py

**resize_image(src_image, scale)**

این تابع یک تصویر یک عدد (برحسب درصد) به بعوان ورودی می‌گیرد و طول و عرض تصویر را با نسبت داده شده تغییر می‌دهد و عکس تغییرسایز داده‌شده را به عنوان خروجی برمی‌گرداند.

**apply_sobel_filter_x(gray_image)**

این تابع یک تصویر سیاه سفید را به عنوان ورودی می‌گیرد و فیلتر sobel_x را برای پیدا کردن مرزهای عمودی روی تصویر اعمال می‌کند و نتیجه را به عنوان خروجی برمی‌گرداند.

**apply_sobel_filter_y(gray_image)**

این تابع یک تصویر سیاه سفید را به عنوان ورودی می‌گیرد و فیلتر sobel_y را برای پیدا کردن مرزهای افقی روی تصویر اعمال می‌کند و نتیجه را به عنوان خروجی برمی‌گرداند.

**get_edges(src_image)**

این تابع یک تصویر را به عنوان ورودی می‌گیرد و بعد از آنکه آن را به تصویر سیاه‌سفید تبدیل با استفاده از توابع apply_sobel_filter_x(gray_image) و apply_sobel_filter_y(gray_image) مرزهای تصویر سیاه‌سفید شده را به دست می‌آورد و به عنوان خروجی برمی‌گرداند.

**initialize_cluster_centers(...)**

این تابع با استفاده از ورودی‌هایی که می‌گیرد مختصات کلاستر سنترها را محاسبه می‌کند، به گونه‌ای که توزیع کلاستر سنترها یونیفرم باشد و فاصله هر کلاستر سنتر با کلاستر سنتر مجاور آن S باشد.

**perturb_cluster_centers(cluster_centers, src_image_edges, indexes)**

این تابع کلاستر سنترها ، تصویر مرزها و ماتریس indexes را به عنوان ورودی می‌گیرد سپس در پنجره‌ای با ابعاد indexes (۱۱ * ۱۱) هر یک از کلاستر سنترها را به گونه‌ای جابه‌جا میکند تا در مختصاتی با کمترین گرایان در این پنجره قرار بگیرند.

**cal_dist(cluster_center, pixel_x, pixel_y, scaled_image, S)**

این تابع یک کلاستر سنتر ، مختصات یک پیکسل و S را به عنوان ورودی می‌گیرد و با توجه به روابط زیر معیاری از میزان فاصله‌ی این پیکسل از آن کلاستر سنتر را در فضای $[x, y, l, a, b]$ محاسبه می‌کند و به عنوان خروجی برمی‌گرداند.

$$
\begin{aligned}
d_{lab} &= \sqrt{(l_k - l_i)^2 + (a_k - a_i)^2 + (b_k - b_i)^2} \\
d_{xy} &= \sqrt{(x_k - x_i)^2 + (y_k - y_i)^2} \\
D_s &= d_{lab} + \frac{m}{S} d_{xy} \ ,
\end{aligned}
$$

**slic(cluster_centers, scaled_image, S, f, data, segments=None)**

این تابع تمامی موارد در قسمت **روند کلی حل.** شده است را پیاده سازی می‌کند. در ابتدا محاسبات به صورت پیمایش روی پیکسل‌های پنجره $2S * 2S$ با فور انجام می‌شد که بسیار از نظر زمانی کند بود. در راستای بهبود سرعت این تابع محسابات به صورت ماتریسی انجام شد که سرعت اجرای کد را حدود ۶۰ برابر بهتر کرد.

**draw_boundaries(segments, scaled_image)**

این تابع سگمنت‌ها و تصویرِ تغییرسایز داده شده را به عنوان ورودی می‌گیرد و با استفاده از تابع find_boundaries از کتابخانه skimage مرزهای سگمنت‌ها را به دست می‌آورد و مقدار intensity پیکسل‌های مرزها ۲۵۵ قرار می‌دهد تا مرز سگمنت‌ها در تصویر مشخص شود.

- **q۳.py**

در این فایل ابتدا تعداد کلاسترها و $scale$ تغییر سایز تصویر با استفاده از تابع get_input از ورودی گرفته می‌شود و به ترتیب در متغیرهای num_of_clusters و scale ذخیره می‌کنیم.در ادامه تصویر slic.jpg را از مسیر EX3_Q3/images/ لود میکنم و فضای رنگی آن را به $LAB$ تبدیل کرده و در متغیر lab_image ذخیره می‌کنیم(البته قبل از تبدیل کردن فضای رنگی، فیلتر bilateral روی تصور اعمال می‌کنیم تا نویز در تصویر نهایی کمتر باشد.) تصویر را در ادامه با scaleای که به عنوان ورودی از کاربرگرفته شده است ریسایز می‌کنیم، کلاستر سنترها را initialize می‌کنیم تا به صورت یونیفرم در تصویر پخش شوند، ماتریس f را به گونه‌ای میسازیم که درایه‌ی $i, j$ آن $i - S, j - S$ باشد (از این ماتریس در تابع slic استفاده میکنیم تا بدون فور بتوانیم پنجره $2S * 2S$ را محاسبه کنیم ) در نهایت هم با استفاده از تابع slic که در فایل utilities.py پیاده‌سازی شده سگمنت‌ها را به دست می‌آوریم و با تابع draw_boundaries مرز سگمنت‌ها را رسم می‌کنیم و نتیجه را با نام test.jpg در مسیر EX3_Q3/results/ ذخیره می‌کنیم.